Dynamic Query Organization and Response Generation in Spoken Dialogue System

Li Fang, Zheng Fang, Wu Wenhu, Huang Yinfei

Center of Speech Technology (CST), State Key Laboratory of Intelligent Technology and Systems

Department of Computer Science and Technology, Tsinghua University, Beijing, 100084, China

Email: lifang@sp.cs.tsinghua.edu.cn

Abstract

EasyNav is a Chinese spoken language dialogue system for campus navigation that we introduced at ICSLP2000 [1]. Recently, we have initiated a significant redesign of some modules to make it resolve some inefficient problem in *EasyNav*, such as analysis of complex sentences, context processing. This Paper describes the query and response generation module of our *EasyNav 2*.0 campus navigation information system. Dynamic query frame is designed according to the requirement of the query module. Not only can this new mechanism deal with most sentence types, but also construct the data retrieval of the query module efficiently. Meanwhile, with the introduction of the query function priority, the performance of the response generation module is improved. The new method is implemented in our campus navigation system and the result of the test is impressive.

Keywords: dynamic query organization, response generation, spoken dialogue system, *EasyNav*

1. Introduction

With the development of the spoken dialog system, how to make it more practical has become one of the main topics in this area. Generally, a typical spoken dialogue system may consist of several components, such as speech recognition, natural language understanding, and dialogue management, speech synthesis. Recent years, there are many research projects devoting into this field all over the world, such as DARPA Communicator in America, ARISE, REWARD and VERBMOBIL in Europe. Also many famous colleges and institutes such as MIT-SLS Lab. CMU-ISL Lab. Lucent-Bell Lab., ATR Lab. in Japan, Erlangen-Nuremberg Univ. in Germany. Some institutes in China such as the Chinese Academy of Sciences [2], The Chinese University of HK, Taiwan Univ. TsingHua Univ., etc., all dedicate a great deal of effort into the research of spoken dialog systems.

A main consideration of spoken dialog system is to facilitate the interactions between humans and computers through natural language, so that humans can make use of extensive resources efficiently by using computers. Spoken dialog system must be natural, intelligent and friendly: (1) Natural: There is no need to set limit upon the utterance，so that user can use natural language during the interaction. (2) Intelligent: System can deal with the problems during the interaction, such as contradiction, ambiguity, deflection from the topic etc. so that the interaction can successfully go on. (3) Friendly: User has high degree of freedom during the interaction and system can give feedbacks to the user at any time. Now, there are several practical spoken dialog systems served as the test bed for the research and development of human language technologies, such as *GALAXY I* and *GALAXY II*, based on which are some task-oriented or domain-dependent applications (e.g., automobile classified ad. [3], restaurant guide and weather information), which symbolize the great achievement in the research of spoken dialog system.

This paper introduces the *EasyNav* 2.0 system — a Chinese spoken dialog system for campus navigation information accessing. Through this system, users can query about a special site, search for some sites that match the request, find out the route to a special site and get other information on campus. Now a prototype of version 2.0 has been finished. Text instead of speech is served as input and the output include both text and an indication on a campus map. Query is limited to information related to the campus of Tsinghua University but query form is unlimited to user. The *EasyNav* system has been served as an exhibition system of our lab, and it proved to be an impressive navigation guide.

2. Architecture Design

The system architecture is shown in Fig. 1. The first four modules process the text input and user intention is extracted and stored into a query frame, which is then passed into query module to search in the campus map database. The searching result is used by response generation module to generate the text output and the map indication.

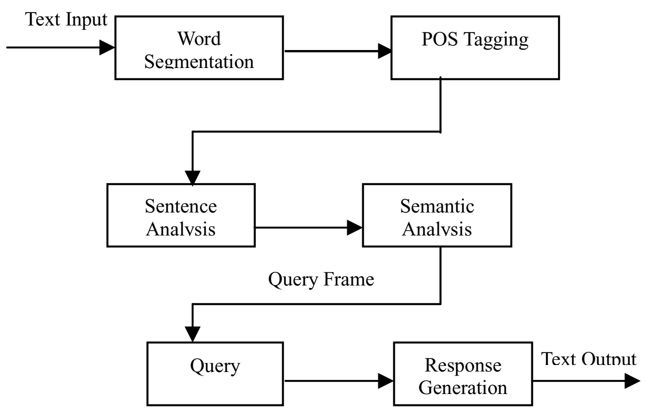

Fig. 1 EasyNav System

Different from the client-server architecture of *GALAXY* [4], *EasyNav* system is designed to run on single computer. More consideration is put on the processing of special syntactic phenomena of Chinese. For examples, word segmentation module deals with the problem that there is no segmentation symbol between Chinese words in a sentence, and semantic analysis module deals with some Chinese idioms and elliptical sentences, and campus map database is designed to be a domain-independent multi-furcated tree structure (Fig. 2) [5], which is powerfully expressive and easy to expand.

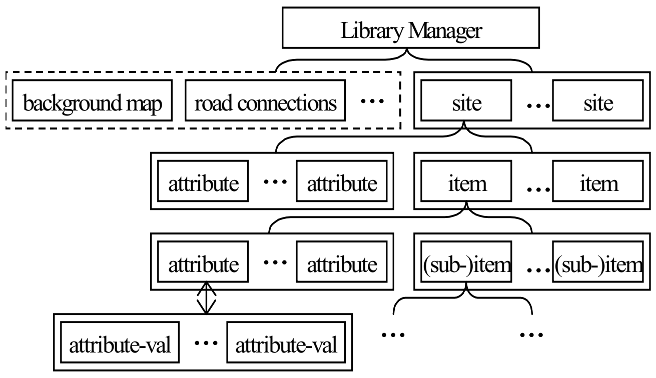

Fig.2 The database structure

3. Problems With Static Query Frame

In many systems, the semantic analysis of the input sentence is often a whole-sentence analysis, which is keyword based and gets information by analyzing the keywords that is recorded in a predetermined table, which will be called static query frame in this paper. The static query frame is passed to query module and the query result is written back to the frame so that the following processes such as response generation and history record management can use it. *EasyNav* 1.0 utilize this method in its implementation. The structure of static query frame is shown in Table 1.

Items in static query frame	Comment
QueryType	Query types, where etc.
AddOnInfo	Additional query info，nearby etc.
ReferenceName	Reference item
FocusName	Focus item
FocusCategory	The category of focus item
Action	Verb
SubObj	Subject and object of an action
AttrName	Attribute name
AttrVal	Attribute value
OriVal	Orientation value
MarchMode	Walk, car etc.
MePos	User position

Table 1 The structure of static query frame

There are several deficiencies in the static query frame. First, because of the limitation of the static query frame, it is difficult for some complex sentences to be properly analyzed. Second, the static query frame does not offer enough instructions for the query module. Moreover, in some worse cases, the information contained in static query frame may be redundant or incomplete (e.g. the relationship among separate parts in a sentence can not be denoted in this way). That makes it necessary for the database query module to commit further semantic analysis which results unclear partition of the modules and hard maintenance.

句式：哪里
(Query Type: Where)
主语：空
(Agent: NULL)
动作：买
(Action: Buy)
宾语：书
(Theme: Book)
一般属性：清华大学出版社
(General Attribute: Tsinghua publishing company)

Fig. 3 Static query frame

Following is an example of the static query frame, “Where can I buy some books published by Tsinghua publishing company?”

In *EasyNav* 1.0, after the processing by the understanding module (the first four modules in Fig. 1), a static frame shown in Fig. 3 can be generated. In this query frame, it is obviously that nothing is in charge of instructing how to search in the database query module. Further more, it is ambiguous whose attribute “Tsinghua publishing company” is. If this query frame is passed into database query module, further semantic analysis must be done.

The static query frame based query has to scan every slot in the frame in order to decide which predefined query procedure to be used according to the Query Type, Subject, Action, Focus Name, and Attribute etc. When there exists an attribute, it is query module's responsibility to make certain whose attribute it is. Although seemingly the static query frame contains every parts of the sentence and is complete, it ignores the relationship among words that should be resolved by the semantic analysis module. Then some special procedure must be defined in the query module to deal with some complex sentences. It challenges the independent rule of modeling and is an obstacle to the further development. So the static query frame, although simple to implement, need be improved.

4. Dynamic Query Frame

4.1 The structure of Dynamic Query Frame

To solve the problem mentioned above, *EasyNav* 2.0 exploits a dynamic query frame, which is bound tightly with the query tasks. This kind of query frame has a nested structure, which contains a lot of frames while each frame contains several slots. It is formed dynamically during the semantic analysis. It contains information about how to organize the query process. All query module needs to do is just to resolve primitive query commands.

The dynamic query frame consists of several frames, and each frame consists of several slots. The slot can be one of three following types: (1) value slot; (2) link slot; (3) function slot. A value slot represents an input word from a user input sentence (e.g. in Fig. 4, [Sub] in frame 5, slot 3). We use string variables store these words and they act as function's parameters in the end. A link slot represents a link to other slot, maybe within the same frame or not (e.g. [Site Name Set] in frame 1, slot 5), and denote that the real value can be gained by going along the link. A function slot represents a database query (e.g. [Site Name Set] in frame 2, slot 0), and the real value can be obtained by query into the database. The function in function slot is one of the primitive query functions, which are designed under carefully investigating the relationship between the structure of the campus map database and the query types practically used. It releases the database query module from the semantic analysis during the query. Until now, we have devised about 30 primitive functions and it seems sufficient for the campus navigation. Table 2 shows the some of these functions.

When passing the dynamic query frame to database query module, the understanding module should indicate a starting frame in order to instruct the query. The recursive traversing is the core technique of designing the query module because of the interlinking of the frames. When query module has traversed all the useful frames from the starting frame, the query task is finished naturally.

It is also very convenient to retrieve and store the context by exploiting the dynamic query frame. Due to the independence of every frame, it needs not to begin the query only from the starting frame. When something must be stored as history information, just query from the appropriate frame and slot to retrieve what is needed.

Function Name	Description ([Parameter]: Return value)
QuerySiteLoc	[Site name]: Site location
QuerySiteNameBySiteCat	[Category]: Site name
QuerySiteNameByAgency	[Agency name]: Site name
QuerySiteNameByAction	[Action]: Site name
QuerySiteNameBySiteOrient	[Reference site], [Orientation]: Site name
QuerySiteLocSet	[Site name set]: Site location set
QueryRoute	[Site name of beginning], [Site name of end]: Route
QueryRouteTime	[Site name of beginning], [Site name of end], [Method]: Time to spend
QueryRouteLong	[Site name of beginning], [Site name of end]: Distance
QueryMatched	[Item1], [Item2]: If identical
SetConfirmItem	
QueryAccord	[Site name], [Attribute name], [Attribute value]: If equal
SetSpecialAttr	[Set conditions]: Sites that match the conditions
SetGeneralAttr	
SetObj	
SetSub	
SetObjectAttr	
SetRange	
SetEvaluation	
GetHere	[NULL]: Current position
GetDefault	[Agency name]: Default site name of the agency
SelectOne	Select one site from site set
SelectSome	Select some sites from site set

Table 2 Primitive functions

4.2 Example of Dynamic Query Frame

In *EasyNav* 2.0, the dynamic query frame is shown in Fig. 4. The semantic analysis of current version is done on the concepts, while that of the previous version is done on the whole sentence. A frame can represent each part of the input sentence according to some semantic rules, and the whole sentence can be represented as a list of frames and the relationship among the parts of the sentence can be represented by the links of these frames. Thus the query frames vary according to different user input, so this kind of query frame is called "Dynamic Query Frame". In Fig. 4 there are example sentence "Where can I buy some books published by Tsinghua publishing company" and the dynamic query frame that formed after the analysis of the sentence. It is clearly that every relationship among the words is delivered by the structural expression. For example, "Tsinghua publishing company" is an attribute of "Publish", which needs a procedure to be judged in the static query frame, becomes very obviously now. Moreover, every place that needs a query function is filled with one of primitive functions, which guide database query module to look up in the database with the direction. There is no need for any predetermined query procedure and every query is determined by the input sentence. In this meaning, the dynamic query frame based query is a kind of dynamic query itself.

```
哪里可以买到清华出版社出版的书
(Where can I buy some books published by Tsinghua publishing
company)
{
      [应答内容]={1}.地点名
      ([Response Content]={1}.Site Name)
      [应答内容]={1}.地点位置
      ([Response Content]={1}. Site Location)
  {1
      [地点位置]=Query 地点位置( [地点名] )
      ([Site Location]=QuerySiteLocation([Site Name]))
      [地点名]=SelectOne( [地点名集合] )
      ([Site Name]=SelectOne([Site Name Set]))
      [地点名集合]={2}.[地点名集合]
      ([Site Name Site]={2}. [Site Name Set])
  }
  {2
      [地点名集合]=Query 地点名( [动作] ) <= {3}.[设置受事]
      ([Site Name Set]=QuerySiteName([Action]<={3}.[SetObj]))
      [动作]=买
      ([Action]=Buy)
  }
  {3
      [设置受事]=Set 受事( [受事] ) <= {4}.[设置物品属性]
      ([SetObj]=SetObj([Obj])<={4}.[Set Object Attr])
      [受事]=书
      ([Obj]=Book)
    }
    {4
      [设置物品属性]=Set 物品属性( [动作] ) <= {5}.[设置施事]
      ([Set Object Attr]=SetObjAttr([Action])<={5}.[SetSub])
      [动作]=出版
       ([Action]=Publish)
    }
    {5
      [设置施事]=Set 施事( [施事] )
       ([SetSub]=SetSub([Sub]))
      [施事]=清华大学出版社
       [Sub]=Tsinghua publishing company
    }
}
```

Fig. 4 Dynamic Query Frame

5. The design of Query Module

As the dynamic query frame provides sufficient information and a starting frame, using recursive technique to traverse all the frames is a natural idea to design the query module. Every level of the recursion deals with the current slot according to three kinds of slots. With a value slot, just return its string value to the caller. With a link slot, take its link destination (denoted as a "frame index" and a "slot index") as the next level of recursion's parameter and call next level. With a function slot, the operation is a little complex. Because a function maybe has some parameters, the first thing to do is retrieve the value of these parameters. In dynamic query frame, the function's parameters are also links except that they can just link to slots within the same frame as the function (e.g. a typical denotation of a parameter is a "slot index"), so similarly as link slot, just recursive call will get the values of parameters. After the retrieval of the parameters, call the query functions respectively to query the database. The symbol "<=" in Fig. 4 represents a precondition. A precondition is actually a link to other slot and will end at one or more functions, so in the same way to carry them out.

Generally, the query conditions of user input operate on the same query object. For example, in sentence "Where is the nearest gate from here", the query conditions are: (Type = Gate) and (Special Attribute = Nearest). These conditions operate on the same query object. From the recursive design we mentioned above, it is obvious that these conditions will appear on different recursive level. In order to optimize the query, it is useful to keep a query result list, which stores the last query result and serves as the searching set during the next query. This mechanism can avoid reiteration on the whole database in our implementation. For example, in Fig. 4, the query order is: QuerySiteName(" buy "), SetObj(" book "), SetObjAttr(" publish "), SetSub(" Tsinghua publishing company "), and these queries operate with AND, so a result list can make the searching set smaller and smaller and retrieve the answer in the end.

Though, there are some occasions that the query conditions operate on different query objects. For example, in sentence "Is the building in front of the library the auditoria" contains a comparison of two different query objects (the building in front of the library, and the auditoria). In this case, the starting frame of the dynamic query frame contains such slot: "[Affirm Result]=QueryAccord()<={1}.[SetAccord], {4}.[SetAccord]", which indicate that there are two query objects in this query frame, so another result list will be needed to deal with the second query object. After the query, the two result lists will be compared to judge if they are identical. In more complex sentence, using similar mechanism to keep N result lists will work.

这附近有没有便宜的食堂？
(Is there any cheap dining room nearby?)

查询函数：Query 类别（"食堂"）
(Query function: QueryCategory("Dining Room"))
Result List：八食堂，七食堂，教工食堂，西餐厅……
(Result List: Dining room 8, Dining room 7, Teacher's dining room, West dining room)
查询函数：Set 一般属性（"便宜"）
(Query function: SetGeneralAttr("Cheap"))
Result List：八食堂，教工食堂
(Result List: Dining room 8, Teacher's dining room)
查询函数：Set 范围（"这里"，"附近"）
(Query function: SetRange("here", "nearby"))
Result List：NULL
(Result List: NULL)

Fig. 5 Make use of priority in response generation

6. Response Generation

The response generation module is in charge of organizing the result retrieved from the database to form the text output and for further processing by the speech synthesizer. For response generation, it is important to be accurate, intelligent and friendly. It must also provide as many as possible information not only in a successful query, but also in a failure. The reason of a failure and the causing function should be reported to make the interaction friendly. On the other hand, the response generation module can direct the query module to avoid some failure causing by too many query conditions.

After the traversing of the whole query frame, if the query is successful, it is convenient to generate proper response with the information contained in the query frame. If the query fails, it is useful to introduce the priority of the functions so that the response will be more intelligence and friendly. A higher priority means it is executed earlier than lower one, so the priority will be shown in the dynamic query frames. The example is shown in Fig. 5. If the priority of the query functions is: QueryCategory > SetGeneralAttr > SetRange, Once the result list is empty, we can try to get rid of current function (e.g. SetRange) in order to get a non-empty result list (e.g. result list: Dining room 8, Teacher's dining room). Thus a successful query is achieved and a response can be "there is no cheap dining room nearby, but Dining room 8 and Teacher's dining room are both cheap dining rooms." With this mechanism, a lot of queries that contain empty result will return some middle results and the originally stiff response can be refined, thus improve the performance of the dialog system.

When applying this mechanism, there exist some difficulties, such as how to produce more natural reply, especially to explain an error. The response generation must know something about the semantic roles of reply words in order to arrange them to be a natural sentence. There are two solutions for the problem.

(1) Cat (not NULL, FALSE): There is no [Cat].

(2) Cat (not NULL, TRUE) AND Evaluation (not NULL, FALSE): There is no [Evaluation] [CAT].

(3) Cat (not NULL, TRUE) AND Evaluation (not NULL, TRUE) AND Range (not NULL, FALSE): There is no [Evaluation] [Cat] [Range].

Fig.6 Example templates

First, by recording the words used in query sentence, we can arrange them to a natural sentence according to a series of templates when there is an error. We record the words in a static query frame-like structure (called Record Structure). Each item in the record represents a part of the input sentence and has two data members: string value and Boolean value. String value contains words itself. Boolean value indicates whether the according function returns successfully. It is obvious that each item has its counterpart function in one query. During each function, we record these two data members of according item. If all the functions success, the reply is easy to generate as before. When some function fails, the query process is interrupted and a special error procedure begins. Every item in the record will be scanned in order to fit for some template. The error explanation will be given according to that template.

For example, if the user input is "Is there any cheap dining room nearby?" In the Record Structure, Item "Cat" contains (dinning room, TRUE), and item "Evaluation" contains (cheap, TRUE), and item "Range" contains (Nearby, FALSE) and all other items contains (NULL, TRUE) after searching in the database by functions. It indicates that function "SetRange" fails. With the example templates shown in Fig.6, the error explanation should be: There is no cheap dinning room nearby. We have implemented this solution and find it can efficiently give the error explanation in most cases.

```
这附近有没有便宜的食堂？
(Is there any cheap dining room nearby?)
(Answer Frame)
{
      [应答内容]= AnswerSiteLocation([地点名]，[地点位置])
      ([Response Content]= AnswerSiteLocation([Site Name], [Site Location]))

}
(Query Frame)
{
      {1
      [地点位置]=Query 地点位置( [地点名] )
      ([Site Location]=QuerySiteLocation([Site Name]))
      [地点名]=SelectOne( [地点名集合] )
      ([Site Name]=SelectOne([Site Name Set]))
      }
                              .
                              .
                              .
}
(Error frame)
{
      [无类别]= ErrorReply([类别])
      [无属性]= ErrorReply([便宜]，[类别])
      [无范围]= ErrorReply([附近]，[便宜]，[类别])
}
```

Fig.7 Answer frame and error frame

The other solution is to ask semantic analysis module generate answer frame and error frame. Answer frame and error frame are similar with query frame in structure and have most of the same advantages.

Answer Frame just deals with the correct reply, that is, none of the function gets an empty result. If there exists an error, for example, no "cheap" attribute for all dining rooms found, then the query procedure is interrupted and a corresponding slot in the error frame is pointed by a link. Error frame contains all error cases that will be encountered during this query. So with very slight modification of the query frame mechanism, we can achieve a more flexible response generation both for correct answer and error explanation. Now we are still working on it.

7. Summery

This paper has focused on the redesign of query and generation module of our *EasyNav* system. This new system exploits query under direction of a list of frames, which are generated dynamically by the semantic understanding. We have found that this mechanism was able to effectively handle the complex query task. Moreover, for friendly response it is useful to introduce the priority of the query functions. With these mechanisms are used, the performance of this dialogue system is improved.

We have tested lots of sentences with varies query types such as Where, Which, How, How long, How far and their sub-types. All the sentences chosen are frequently used during campus navigation. Almost all sentences can be analyzed by our system and obtain satisfied replies. It is a strong proof for the success of our system .

Although there are unrecognized sentences, some of them are due to the insufficient lexicon, which can be expanded gradually. The error explanation still need to be refined in order to produce more natural reply. In the near future, with the research on domain independent, this system will have more practical applications.

References

[1] Huang Y.F, “Language Understanding Component for Chinese Dialogue System”, *ICSLP’2000*, pp. 1053-1056, Beijing, China, Oct.2000

[2] Chao HUANG, Peng XU, et al. “LODESTAR: A Mandarin Spoken Dialogue System for Travel Information Retrieval” *EUROPEAN CONFERENCE ON SPEECH COMMUNICATION AND TECHNOLOGY*, EuroSpeech, 1999, v3, p 1159-1162

[3] DAI L.R., “The feature of human-computer dialogue system and the design of system”, (in Chinese), *Journal of Anhui University*, December, 1997

[4] S. Sneff, E. Hurley, R. Lau, C. Pao, P. Schmid, and V. Zue, “Galaxy-II: A Reference Architecture for Conversational System Development”, *ICSLP’98*, pp. 931-934, Sydney, Australia, December, 1998

[5] Yan P.J., Zheng F., “Word-class stochastic model and knowledge representation in a spoken language dialogue system”, (in Chinese), Journal of Tsinghua University (Science and Technology), 2001, Vol.41, No.1, p. 69-72